AMENDMENTS TO THE CLAIMS

This listing of claims replaces all prior versions, and listings, of claims in the application:

1. (Currently Amended) A method for producing, for a target computer architecture and a program fragment, a near-optimal code sequence for executing the program fragment on the target computer, comprising:

repeatedly invoking an automatic theorem prover for plural cycle budgets to

determine a minimum cycle budget that is the lowest of any cycle budget K for which a formalized mathematical conjecture that no code sequence for the target computer architecture executes the program fragment within the cycle budget K is unprovable by the automatic theorem prover, and

extract the near optimal code sequence from a counterexample implicit in the failed proof of the formalized mathematical conjecture for the minimum cycle budget.

2. (Currently Amended) The method of claim 1, wherein the automatic theorem prover is two-phased, the two phases ~~being~~ including

~~a matcher for~~ instantiating facts by a matcher about machine operations that are computable by a machine with the target computer architecture and facts about non-machine operations, followed by

a boolean satisfiability search.

3. (Original) The method of claim 1, wherein the program fragment specifies a vector of expressions to be computed together with one or more of

a vector of target destinations into which the values of the expressions are to be placed, and

a guard and label pair, the guard being a given boolean expression that determines whether the program fragment is to be executed as described or whether, instead, control is to be transferred to the label.

4. (Original) The method of claim 1, wherein, during the invocations of the automatic theorem prover, the minimum number of machine cycles for each successive invocation is set to a value so as to bisect the interval of remaining possible values of the minimum number of machine cycles.

5. (Original) The method of claim 2, wherein the instantiated facts from the matcher are asserted into an e-graph which is formed from a term graph augmented by an equivalent relation connecting terms known to be equal.

6. (Original) The method of claim 2, wherein the satisfiability search operates on a collection of boolean unknowns that encode a set of conjectured code sequences for a machine with the target computer architecture, each of these code sequences being defined in terms of a set of machine operations initiated in each cycle.

7. (Original) The method of claim 6, wherein the instantiated facts from the matcher are asserted into an e-graph which is formed from a term graph augmented by an equivalent relation connecting terms known to be equal, and wherein the encoding is performed such that, for each term of the e-graph and each cycle i of the minimum number of machine cycles for a particular invocation, there is a particular boolean unknown that indicates whether the conjectured code sequence performs a computation of the root operation of the term during cycle i.

8. (Currently Amended) The method of claim 6, wherein the boolean ~~unknown can~~ unknowns encode ~~alternative~~ boolean constraints suitable for the target computer architecture.

9. (Currently Amended) A method for producing, for a target computer architecture and a program fragment, a near-optimal code sequence for executing the program fragment on the target computer, comprising:

repeatedly invoking an automatic theorem prover to prove unsatisfiable a formalized mathematical conjecture that, for a cycle budget K, no code sequence for the target computer architecture executes the program fragment within that cycle budget K,

wherein if the proof fails, a K-cycled program computing the program fragment is embedded in the failed proof,

wherein the near-optimal code sequence is found, and the invocation need not be repeated, when it is established that both the K-cycled program computes the program fragment and a cycle budget K-1 is insufficient in that the cycle budget K is minimum, the K-cycled program being extracted as the near-optimal code sequence, and

wherein, if the near-optimal code sequence is not found in a present invocation, for a next revocation of the automatic theorem prover if the proof succeeds the ~~budget~~ cycle budget K is doubled (K:=K*2) and if the proof fails the ~~budget~~ cycle budget is bisected (K:=K/2) and a new K-cycled program computing the program fragment that is embedded in the failed proof is extracted.

10. (Original) The method of claim 9, wherein the program fragment is presented to the automatic theorem prover as a set of guarded multi-assignments each including a guard and a multi-assignment that can be performed only when its respective guard is true.

11. (Currently Amended) The method of claim 10, wherein the set of guarded multi-assignments is compiled by instantiating universal facts about ~~relevant and useful~~ operators including machine and non-machine terms, wherein each instance of ~~relevant and useful~~ operators provides a way for computing a corresponding multi-assignment.

12. (Original) The method of claim 11, wherein the ways for computing the multi-assignments are encoded in a graph.

13. (Original) The method of claim 12, wherein the graph is an equivalence graph (e-graph) formed as a directed acyclic graph.

14. (Currently Amended) The method of claim [[11]] 12, wherein the graph is transformed in the presence of equalities between nodes.

15. (Currently Amended) The method of claim [[11]] 12, wherein the graph is submitted for the extraction of the near optimal code sequence, the extraction using~~, in addition,~~ a description of the target computer architecture for formulating a boolean satisfiability problem a solution of which is found for the minimum cycle budget K via a satisfiability search.

16. (Original) The method of claim 12, wherein for a multi-assignment of the size n, an e-graph with a size order of n represents 2^n distinct ways of computing the multi-assignment.

17. (Original) The method of claim 9, wherein the extraction of the near optimal code sequence is done from a formulation of a boolean satisfiability problem using a set of boolean unknowns that are one-to-one corresponding to a solution of the boolean satisfiability problem, the solution corresponding to a budget-cycle machine program where the budget is the minimum cycle budget K.

18. – 19. (Cancelled)

20. (Original) The method of claim 1 wherein the automatic theorem prover performs refutation-based automatic theorem proving.

21. (Original) The method of claim 9 wherein the automatic theorem prover performs refutation-based automatic theorem proving.

22. (Currently Amended) A method for automatic generation of a near-optimal code sequence for execution on a computer, comprising:

~~repeatedly~~ applying automatic theorem-proving to a code sequence generator, including

introducing a multi-assignment to the code sequence generator,

~~performing matching~~ producing, by the code sequence generator based on the multi-assignment, a number of possible plans for creating the near-optimal code sequence, and

performing, by the code sequence generator, planning with a satisfiability search to select~~, wherein the matching produces a number of possible plans for creating the near-optimal code sequence, and wherein the planning selects~~ an optimal plan from among the possible plans ~~discovered via the matching, thereby~~ for automatically producing the near-optimal code sequence, wherein performing the planning with the satisfiability search is repeated a plurality of times for plural machine cycle budgets to find the optimal plan associated with a predetermined machine cycle budget.

23. (Original) A method as in claim 22, wherein the multi-assignment includes goal terms that specify what result the near-optimal code sequence is expected to produce, and wherein the applying automatic theorem proving further includes initializing a term graph with the goal terms whereby nodes of the term graph receive the goal terms.

24. (Currently Amended) A method as in claim 23, ~~wherein the matching includes~~ further comprising:

introducing instances of universal facts that are relevant to the near-optimal code sequence, and

augmenting the term graph with equivalence relations between the goal terms and corresponding instances of the universal facts by matching the universal facts against the term graph.

25. (Currently Amended) A method as in claim [[22]] 23, wherein values of the goal terms are computed into registers of the computer, the registers being specified in the multi-assignment.

26. (Original) A method as in claim 24, wherein the term graph is augmented by the equivalence relations on its nodes to produce an equivalence graph (e-graph).

27. (Currently Amended) A method as in claim 26, ~~wherein the matching transforms~~ further comprising transforming the e-graph into a transformed e-graph that is [[then]] provided to the planning [[for]] with the satisfiability search.

28. (Original) A method as in claim 24, wherein the satisfiability search produces the near-optimal code sequence for achieving values corresponding to the goal terms.

29. (Currently Amended) A method as in claim [[28]] 23, wherein the near-optimal code sequence is created from the term graph by iteratively solving a satisfiability problem with ~~various~~ the machine cycle budgets until an optimal code sequence is found.

30. (Original) A method as in claim 24, wherein the universal facts are available in a file and are introduced as an input to the code sequence generator so that the universal facts can be changed without changing the code sequence generator.

31. (Original) A method as in claim 24, wherein the universal facts express properties of operators in the goal terms.

32. (Original) A method as in claim 25, wherein the term graph is initialized with node terms representing the goal terms.

33. (Cancelled)

34. (Currently Amended) A method as in claim [[33]] 22, wherein the predetermined machine cycle budget is a minimal ~~number of cycles~~ machine cycle budget.

35. (Original) A method as in claim 22, wherein the satisfiability search is a goal-directed search.

36. (Currently Amended) A code sequence generation tool for automatic generation of a near-optimal code sequence, comprising:

an input capable of receiving a multi-assignment;

a matcher responsive to the multi-assignment and producing, via matching of the multi-assignment and facts regarding operators computable in a computer, a number of possible plans for creating the near-optimal code sequence; and

a planner configured to select via a satisfiability search an optimal plan from among the possible plans ~~discovered~~ produced by the matcher, the optimal plan corresponding to the near-optimal code sequence,

wherein the code sequence generation tool is configured to invoke the matcher and the planner thereby implementing [[an]] automatic theorem-proving for automatically generating the near-optimal code sequence.

37. (Original) A code sequence generation tool as in claim 36 being further configured for producing the optimal code sequence using a goal-oriented, cycle budget limited code sequence in generating the near-optimal code sequence.

38. (Original) A code sequence generation tool as in claim 36 wherein the planner includes a constraint generator and a solver, the code sequence generation tool further comprising an input configured for introducing architectural constraints to the constraint generator which the constraint generator uses in creating a set of boolean unknowns for the solver.

39. (Currently Amended) A code sequence generation tool for automatic generation of a near-optimal code sequence, comprising:

an input capable of receiving a multi-assignment;

matching means responsive to the multi-assignment and producing, via matching of the multi-assignment and facts regarding operators computable in a computer, a number of possible plans for creating the near-optimal code sequence; and

planning means configured to select via a satisfiability search an optimal plan from among the possible plans ~~discovered~~ produced by the matching means, the optimal plan corresponding to the near-optimal code sequence,

wherein the code sequence generation tool is configured to invoke the matching means and the planning means thereby implementing [[an]] automatic theorem-proving for automatically generating the near-optimal code sequence.

40. (Cancelled)

41. (New) The method of claim 22, further comprising executing the code sequence generator as a computer-executed code sequence generator.

42. (New) The code sequence generation tool of claim 36, wherein the planner is invocable a plurality of times for plural machine cycle budgets, the planner to select the optimal plan associated with a minimum machine cycle budget from among the plural machine cycle budgets.

43. (New) The code sequence generation tool of claim 39, wherein the planning means is invocable a plurality of times for plural machine cycle budgets, the planner to select the optimal plan associated with a minimum machine cycle budget from among the machine cycle budgets.

44. (New) A method of producing a near-optimal code sequence for at least a fragment of a program, comprising:

inputting expressions corresponding to the fragment of the program to a computer-executable code sequence generator;

generating, by the code sequence generator based on the input expressions and facts regarding operators computable in a computer, a data structure representing plural ways of computing the expressions; and

performing a satisfiability search by the code sequence generator to select one of the ways as an optimal solution associated with a minimum machine cycle budget, the optimal solution corresponding to the near-optimal code sequence.

45. (New) The method of claim 44, wherein performing the satisfiability search is repeated plural times for plural machine cycle budgets.

46. (New) A computer-readable medium embodying computer program code configured to cause a computer to generate a near-optimal code sequence for at least a fragment of a program, comprising:

inputting expressions corresponding to the fragment of the program to a code sequence generator;

generating, by the code sequence generator based on the input expressions and facts regarding operators computable in a computer, a data structure representing plural ways of computing the expressions; and

performing a satisfiability search by the code sequence generator to select one of the ways as an optimal solution associated with a minimum machine cycle budget, the optimal solution corresponding to the near-optimal code sequence.

47. (New) The computer-readable medium of claim 46, wherein performing the satisfiability search is repeated plural times for plural machine cycle budgets.